Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L9 and (instance near type)

Display: 50 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Friday, March 04, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L10	L9 and (instance near type)	17	L10
L9	L8 and relationship	33	L9
L8	L7 and instance	34	L8
L7	L6 and schemas	34	L7
L6	L5 and synchroniz$	50	L6
L5	l4 and (file near system$)	103	L5
L4	L3 and (application near program$)	260	L4
L3	L2 and (data near store)	410	L3
L2	L1 and (database near engine)	929	L2
L1	707/$.ccls.	25377	L1

END OF SEARCH HISTORY

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 2 of 2 returned.

☐ 1. Document ID: US 6199195 B1

Using default format because multiple data bases are involved.

L4: Entry 1 of 2 File: USPT Mar 6, 2001

US-PAT-NO: 6199195
DOCUMENT-IDENTIFIER: US 6199195 B1

TITLE: Automatically generated objects within extensible object frameworks and links to enterprise resources

DATE-ISSUED: March 6, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Goodwin; Richard Glenn	Oceanside	CA		
Farrar; Michael Andrew	San Diego	CA		
Messina; Marvin	San Diego	CA		
Steele; Jason	Lakeside	CA		

US-CL-CURRENT: 717/104; 707/100, 717/108

Full | Title | Citation | Front | Review | Classification | Date | Reference | [illegible] | Claims | KWIC | Draw. Desc | Image

☐ 2. Document ID: WO 200104726 A2, AU 200059248 A, US 6199195 B1

L4: Entry 2 of 2 File: DWPI Jan 18, 2001

DERWENT-ACC-NO: 2001-521269
DERWENT-WEEK: 200233

TITLE: Automatic object oriented source code generation method for business framework, involves generating source code as function of unified model and templates which defines service within framework

INVENTOR: FARRAR, M A; GOODWIN, R G ; MESSINA, M ; STEELE, J

PRIORITY-DATA: 1999US-0350406 (July 8, 1999)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
WO 200104726 A2	January 18, 2001	E	046	G06F000/00
AU 200059248 A	January 30, 2001		000	G06F000/00
US 6199195 B1	March 6, 2001		000	G06F009/45

INT-CL (IPC): G06 F 0/00; G06 F 9/45

ABSTRACTED-PUB-NO: US 6199195B
BASIC-ABSTRACT:

NOVELTY - The logic model generated using modeling tool are translated into corresponding unified model. A system definition comprising templates which defines service within framework is generated. A source code object is generated as a function of unified models and templates.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for source code object generating system.

USE - For generating source code for object oriented application in complex business framework, links to enterprise resources.

ADVANTAGE - Approach is provided for automatically generating source code within a complex business framework and generates business object with all implemented behavior within a composed object service framework.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of computer system.
ABSTRACTED-PUB-NO:

WO 200104726A EQUIVALENT-ABSTRACTS:

NOVELTY - The logic model generated using modeling tool are translated into corresponding unified model. A system definition comprising templates which defines service within framework is generated. A source code object is generated as a function of unified models and templates.

DETAILED DESCRIPTION - An INDEPENDENT CLAIM is also included for source code object generating system.

USE - For generating source code for object oriented application in complex business framework, links to enterprise resources.

ADVANTAGE - Approach is provided for automatically generating source code within a complex business framework and generates business object with all implemented behavior within a composed object service framework.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of computer system.

Term	Documents
"6199195"	3
6199195S	0
"6199195".PN..PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	2
(6199195.PN.).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	2

L10: Entry 17 of 17 File: USPT Jan 29, 2002

DOCUMENT-IDENTIFIER: US 6343287 B1
TITLE: External data store link for a profile service

Abstract Text (1):
A profile service, instance is linked to a plurality of external data stores. Each external data store is associated with a predefined data store connector class that describes a connector object that establishes a link and provides methods to query the associated data store. An external data store profile is created in the profile service that names the connector class. An external data store reference object is created in the profile service instance that identifies the external data store profile and a number of parameters that specify particular data desired from the external data store. A profile within the profile service instance includes an attribute that names the data store reference object. When the attribute is evaluated, the data store reference object is instantiated, optionally using parameters specified at runtime, and passed as a parameter to an instance of the data store connector class identified by the external data store profile. The external data store connector instance applies the query methods to retrieve the desired data and return the desired data to the data store reference object. The profile service instance uses the returned data as the value of the attribute.

Brief Summary Text (10):
Existing mechanisms tend to focus on a single type of profile information, user information or application information or hardware information. Also, because these mechanisms are very application specific they limit the number and type of attributes that can be retained. Further, the profile information is isolated and fails to indicate any hierarchical or relational order to the attributes. For example, it may be desirable that a user group is required to store all files created using a particular application suite to a specific file server. Existing systems, if such a service is available at all, must duplicate profile information in each application program merely to implement the required file storage location preference. Storage location direction based on a user-by-user or user group basis is difficult to implement and may in fact require a shell application running on top of the application suite. Even then, the system is not extensible to access, retrieve, and use profile information for a new user that has not used a particular machine before.

Brief Summary Text (12):
An example of the inefficiencies of these types of systems is found in the Windows 95 registry file that holds profile information but has an acknowledged tendency to bloat over time with duplicative and unused data. Moreover, the registry file in such systems is so closely tied to a particular machine and instance of an operating system that it cannot be remotely accessed and used to configure other computers or devices. Hence, these systems are not generally extensible to manage multiple types of profile information using a single mechanism. A need exists for profile information that is readily accessible to all machines coupled to a network and to machines accessing the network through remote access mechanisms.

Brief Summary Text (17):
Directory software tends to be special purpose to serve the needs of a defined set of users to access information about and stored in a defined set of data store mechanisms. For example, a DOS file system (i.e., a directory of filename:physical location information) is written to be accessible only by a particular operating system (e.g., DOS, Windows, Unix, and the like). Hence, the file system information is not accessible to computers running other operating systems. Similarly, a file system cannot be amended to serve as a directory for other types of devices (e.g., an email directory). Moreover, the functionality of a file system is rigidly fixed and is not readily extended to provide new functionality such as authentication,

replication, file system logging, and the like. These types of changes require rewrite and recompile of the file system software. A need exists for a directory system that is flexible and adaptable to service a variety of user entities, store directory information about a variety of objects, and incorporate a variety of functionality at runtime.

Brief Summary Text (27):
An LDAP directory can be distributed among many servers, and each server can have a replicated version of the total directory that is synchronized periodically. When an LDAP server receives a request from a user, it takes responsibility for the request, passing it to other DSAs as necessary, but nevertheless ensuring a single coordinated response for the user.

Brief Summary Text (29):
Meta-directories are a partial solution that provide a directory integration to unify and centrally manage disparate directories within an enterprise. However, existing solutions are not sufficiently extensible to account for the wide variety and continuously changing set of resources for which directory information is desirable. In particular, links to external data store devices are difficult to configure and limited in variety. A meta-directory manufacturer provides a limited set of directory services (e.g., LDAP, X.500, and the like) and the user is limited to those provided services. As a result users cannot link to new data store services that become available.

Brief Summary Text (30):
Moreover, in the past, meta-directory technology has not been used to catalog meta-data of sufficiently general nature to meet the needs of a dynamically growing and changing distributed computing environment. Also, meta-directory software continues to have the disadvantages of being written to support a specific, narrow set of users working on software/hardware platforms in a manner that provides a defined, non-extensible set of functionality. What is needed is a service architecture that provides directory integration together with an ability to add links to new external data store mechanisms specified at runtime.

Brief Summary Text (32):
Briefly stated, the present invention involves a mechanism, method, and computer program product for linking a profile service instance to a plurality of external data stores. Each external data store is associated with a predefined data store connector class that describes a connector object that establishes a link and provides methods to query the associated data store. An external data store profile is created in the profile service that names the connector class. An external data store reference object is created in the profile service instance that identifies the external data store profile and a number of parameters that specify particular data desired from the external data store. A profile within the profile service instance includes an attribute that names the data store reference object. When the attribute is evaluated, the data store reference object is instantiated, optionally using parameters specified at runtime, and passed as a parameter to an instance of the data store connector class identified by the external data store profile. The external data store connector instance applies the query methods to retrieve the desired data and return the desired data to the data store reference object. The profile service instance uses the returned data as the value of the attribute.

Drawing Description Text (7):
FIG. 6 illustrates an exemplary external data store profile in accordance with the present invention;

Drawing Description Text (8):
FIG. 7A through FIG. 7D show class components and relationships between classes used in an exemplary embodiment of the present invention;

Detailed Description Text (2):
The present invention is described in terms of a specific embodiment involving a method and system for implementing a link between a profile service an external data store. This feature turns a profile service into a functional meta-directory enabling profile information to be distributed throughout a distributed computing system. More broadly, however, the external data store link mechanism in accordance with the present invention may be applied in a wide variety of client/server applications. Accordingly, the specific examples and implementations described herein are not to be construed as limitations of the invention as claimed unless specifically

noted otherwise.

Detailed Description Text (5):
The present invention primarily involves a methodology and system that can be implemented as computer program products to support integration of an arbitrary number of external data stores into a service application such as a profile service. The preferred implementations use plug-in service provider interfaces to implement data store specific access protocols for data storage mechanisms including naming and directory services for data storage mechanisms. In this manner, the core profile engine is readily extended to support new data store mechanisms, using new hardware devices and network configurations to provide meta-directory services for a heterogeneous set of data store mechanisms.

Detailed Description Text (7):
It is contemplated that the present invention will be particularly useful in environments that need to be extensible and support a wide variety of data store mechanisms. The present invention provides a method and mechanism for defining new data stores, and so can be adapted to new data store mechanisms that were unavailable or unknown when the service is first implemented. Also, the system of the preferred implementation is optimized to store and make available relatively compact units of data that serve to configure devices and computer environments rather than operational or analytical data upon which the computer environment may operate at runtime. Hence, the present invention is best used when it stores and retrieves data that is frequently searched and retrieved, but infrequently changed.

Detailed Description Text (11):
Data store A data storage mechanism such as magnetic, optical, magneto-optical or solid state memory. A data store is a logical entity that may comprise one or more physical storage devices such as a RAID storage mechanism. The term data store also includes software applications that act as a shell or proxy for a physical data storage device such as a data access service or a database management software application.

Detailed Description Text (12):
Dedicated data store A data store mechanism that is exclusively accessible to the profile service so that no other applications can alter data within the dedicated data store.

Detailed Description Text (13):
External data store A data store mechanism that is coupled to the profile service through a data communication link. An external data store may be local (i.e., closely coupled through a data bus) or remote (i.e., coupled through a network connection) with respect to the profile service instance.

Detailed Description Text (30):
The environment of FIG. 2 includes two profile services instances 201 and 202. Each profile service instance is implemented in a separate geographic environment (e.g., a LAN or standalone environment) as suggested by the dashed vertical lines in FIG. 2. The local environments are coupled by an available WAN connection provided by, for example, a continuous connection, an on-demand connections, and/or multiplexed connections.

Detailed Description Text (31):
A client application 211 accesses the profile service instance 201 to request profile services. The profile service instance 201, performs the requested service using a virtual profile data store comprising dedicated data store 206, local external data store 208 and remote external data store 209. Profile service instances 201 and 202 are each associated with a dedicated data store 206 and 207 respectively. The dedicated data stores 206 and 207 are local to their associated profile service instance and are not used by other applications. Dedicated data store 206 and 207 may be implemented using any available persistent storage device such as a magnetic, optical, or magneto-optical disk, solid state memory, and the like. Dedicated data store 206 and 207 may use only a portion of the physical storage device upon which they are implemented. For example, data store 206 may be a single volume or file on a multi-volume storage device. Likewise, dedicated data store 206 and 207 may each comprise multiple physical storage devices that cooperate through, for example, a RAID controller or file system software.

Detailed Description Text (32):

In an exemplary implementation, profile service instances include a build-in adapter for coupling to their associated dedicated data store. The built-in adapter may be implemented using, for example, a lightweight directory access protocol (LDAP) that provides an industry standard directory access mechanism. Other directory access protocols including industry standardized and proprietary protocols may be equivalently substituted in particular applications. A feature of the present invention is that some or all of the contents of a dedicated data store are replicated across each dedicated data store of each profile service instance in a given system. It is contemplated that not all of the data need be replicated as some profile data will be of a nature that will only be used in a given geographic area and replication may be wasteful. For example, if it is known that client application 214 never requests profile services through profile service instance 201, any profile information held in dedicated data store 207 about client application 214 need not be replicated in dedicated data store 206. The degree and manner of replication is determined to meet the needs of a particular application.

Detailed Description Text (33):
"User entities" such as client software and/or hardware use the profile service by establishing a runtime binding to a profile service instance. In FIG. 2, client applications 211, 212, 213 and 214 and application 204 represent user entities. Client application 212 is an indirect user of profile service instance 202 because it accesses through the domain-specific application 204. Domain specific application 204 is essentially an adapter or shell that provides accessibility when, for example, client application 212 is unable to communicate with an external service.

Detailed Description Text (34):
Each profile service instance 201 and 202 include one or more plug-in remote protocol adapters in addition to any built-in protocol adapters. Each remote protocol adapter implements a transport protocol supporting communication with a client 211-214) and a particular communications network used by the client. For example, remote protocol adapters may implement hypertext transfer protocol (HTTP) with embedded extensible markup language (XML) documents, HTTP with hypertext markup language (HTML) forms, remote method invocation (RMI), common object request broker (CORBA), and the like. It is contemplated that other transport mechanisms may be useful in particular applications such as transport mechanisms specified for fibre channel fabrics as well as proprietary transport protocols. The markup language document is used to encode commands and control information in a declarative fashion in a readily transportable fashion. Accordingly, any available encoding format that is readily transportable using the available transport mechanism (e.g., HTTP) is suitable. These and other implementations are considered equivalent to the specific embodiments disclosed herein unless specifically indicated otherwise.

Detailed Description Text (35):
Important functions of a protocol adapter include providing a data transport mechanism that is compatible with the associated client and with the physical communication link between the client and the profile service instance. Further, where the data transport mechanism requires, the protocol adapter must translate messages from and to the client application into a form that can be embedded in the data transport mechanism. In addition to plug-in protocol adapters, one or more protocol adapters may be built in to the profile service itself.

Detailed Description Text (36):
Each profile service instance 201 and 202 include plug-in interfaces for coupling to external data store mechanisms. As shown in FIG. 2, profile service instance 201 couples to a local external data store 208 and a remote external data store 209. In operation each profile service 201 and 202 make a runtime binding to an arbitrary number of storage provider plug-ins to make the necessary connections to store and retrieved data from external storage devices. External storage devices 208, 209, 210 and 217 may be accessed using any available storage access mechanisms including X.500, LDAP, Novell directory service (NDS), network file system (NFS), network information system (NIS), remote method invocation (RMI), common object request broker (CORBA) and the like. By providing an appropriate plug-in, new directory services that have not been defined when the core profile service is written can be integrated into the system in accordance with the present invention.

Detailed Description Text (37):
As illustrated by the instance in region 215, multiple client applications 212-214 may access a

single profile service instance. Typically, a client application would attempt to access the profile service instance in the same geographic area, however, one may not be available as in the case of mobile users. Because the profile service instance can plug-in remote protocol adapters as needed to support a communication link, the client applications need not be using the same hardware or software platform, and may be using different data transport protocols to access profile service instance 202. Similarly, a single profile service instance can attach to a variety of heterogeneous data store devices simultaneously.

Detailed Description Text (38):
As shown in geographic region 225, remote data service applications such as service 216 can also be attached using storage provider plug-ins. A data service application may be implement a comparatively simple operation such as a network file system or more complex such as a meta-directory, an on-line data service such as LEXIS or DIALOG commercial on-line data services, an internet search engine, or a database front end such as SQL database management software.

Detailed Description Text (39):
As suggested by link 226, multiple profile service instances can be federated to establish a single logical profile database by setting configuration parameters on the desired instance (e.g., instances 201 and 202 in FIG. 2). Collaboration is implemented utilizing a combination of profile and field level identifiers and request forwarding between instances of the profiling service over link 226. When two profile instances are linked, the profile information stored in the dedicated databases 206 and 207 become available to each of the instances. This creates a potential for conflict if a profile service attempting to access a profile or attribute within a profile cannot distinguish between two profiles or between two or more attributes of a profile. In accordance with the present invention, each profile and each attribute is marked with a resource identifier. Previously unconnected profiling service instances can be connected with virtually no risk of data conflicts by leveraging the resource identifiers built-into the core profiling service.

Detailed Description Text (40):
FIG. 3 shows a more detailed example of an implementation of the present invention. The components include a core profile engine 301 that is accessed by a client application 302 through a profile services application programming interface (API) 303. API 303 implements within itself or attaches to one or more protocol adapters 304. Client applications 302 that have a corresponding interface to one of protocol adapters 304 send and receive messages through API 303 to core profile engine 301. The messages enable client application 302 to send data and commands to request profile services from core profile engine 301.

Detailed Description Text (42):
Core profile engine 301 responds to the client request messages by passing the message to an appropriate method to execute requested functions on virtual profile data store 314. Core profile engine 301 comprises a profile manager object and a plurality of profile objects that are described in greater detail with reference to FIG. 5A and FIG. 5B. Typically the core profile service will respond to a request message by accessing profile information from its dedicated data store or an external data store (shown in FIG. 2) and generating a response message. The response message is sent back through API 303 to the appropriate protocol adapter 304 (or built-in adapter 305) to the requesting client application 302.

Detailed Description Text (43):
In the implementation of FIG. 3, core profiling engine 301 includes a built-in interface for attaching to data storage devices. In the particular example of FIG. 3, Java.TM. naming and directory interface.TM. (JNDI) is used. JNDI is a commercially available naming and directory interface that includes a pluggable service provider interface (SPIs). JNDI is essentially an API that provides naming and directory functionality to applications written in a Java programming environment. Java and Java Naming and Directory Interface are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. JNDI is defined to be independent of any specific directory service implementation. Hence, a variety of directories including legacy, emerging, and already deployed directories can be accessed in a common manner. In operation core profile engine 301 causes JNDI to create a transparent runtime binding to naming and directory service such as an X.500 or LDAP data store as shown in FIG. 3.

Detailed Description Text (44):

It is contemplated that instead of or in addition to JNDI the present invention may also incorporate a built-in interface to support directory access to its associated dedicated data store 307. Because dedicated data store 309 is not accessed by other applications, a compiled, built-in interface may be more appropriate and efficient than a plug-in interface. In the example of FIG. 3, built-in LDAP module 308 is used to access dedicated data store 309. However, the JNDI layer provides flexibility in the choice of the mechanism used to implement dedicated data store 309 as the LDAP module 308 is readily replaced by any of a wide variety of available modules.

Detailed Description Text (45):
Virtual profile data store 314 may comprise a single data storage device, but more often comprises a plurality of disparate, heterogeneous data storage devices. The specific example of FIG. 3 includes an LDAP data store, 307, X.500 data store 311, and a relational database 306 accessed through a database application such as a structured query language (SQL) server 310. As noted above, virtual profile data store 314 may also include flat data file(s), object oriented database(s) and the like. Virtual data store 314 includes a dynamically changing number of data store devices as data store mechanisms can be added, changed, modified and deleted by modifications to the associated adapter module.

Detailed Description Text (49):
Yet another example of a plug-in module is event logging module 321 that functions to record events that occur in the profiling service. File systems often are implemented with logging functions to enable the system to recreate transactions in the event of hardware failure. Logging module 321 is similar to such a logging function, however, is not limited to file-system type logging. Instead, any profile information or operation within profile service 201 (shown in FIG. 2) may be logged.

Detailed Description Text (52):
Replication plug-in 324 implements functionality required to replicate the contents of dedicated data store 206 and 207 shown in FIG. 2. As noted hereinbefore, users may wish to customize the replication methodology by specifying only a portion of the dedicated database that is replicated. Moreover, user's may wish to specify the frequency and timing of replication events to balance the need for replication against the cost of transporting data between replicas. Replication plug-in provides this customizable functionality.

Detailed Description Text (53):
The present invention is intended to integrate several physical data stores into a single, distributed logical data store of reference information. The profile service in accordance with the present invention provides a high-speed mechanism to lookup, structure and store key/value pairs stored in data structures called profiles. These key/value pairs represent information about "entities" such as application software, users, hardware devices, and the like.

Detailed Description Text (54):
As used herein, a "profile" is a data object containing a set of key/value pairs, such as profile 400 shown in FIG. 4. Each key/value pair is referred to as an "attribute" such as attribute 412 in FIG. 4. The value associated with a given key may be either a primitive value (e.g., a numeric value, string value, logical value, and the like) as illustrated at 401, another profile as illustrated at 413, or an external data store reference as shown at 402, When the value is another profile the value is referred to as a "subprofile or binding". An individual profile in data structure 400 comprises 0 to n attributes and 0 to n subprofile bindings 402 where "n" is an arbitrary value selected to meet the needs of a particular application. In the example of FIG. 4, profile names are enclosed by brackets [] and attributes 401 are represented as key=value pairs.

Detailed Description Text (59):
DemographicList and PostalworkList in FIG. 4 are examples of another special profile type called a "profilelist". A profilelist relates multiple profiles of the same type to a single parent. This is done by binding a single profile list (e.g., postalworklist) to the parent (e.g., Contact) with multiple member profiles (e.g., "gb" and "us" in FIG. 4) are added to the list. While ProfileLists are serve to collect profiles of the same type, it is contemplated that they may also be used as a convenient collection object for subprofiles of different types. It is important to note that ~~profilelist~~ member profiles may not contain any attributes and do not support the concept of a schema. As a result of this limitation, several profile

operations are not supported by ProfileLists.

Detailed Description Text (61):
FIG. 5 shows a simplified data structure within a external data reference object 501 that is the value of attribute 402 in FIG. 4. As suggested by the structure of FIG. 5, external data reference object 501 may be implemented as a profile wrapped in an object defined by an external data reference class 721 shown in FIG. 7C. The attributes of object 501 may be stored for example, in dedicated data store 206 or 207. Significant attributes of object 501 include a "data store profile" attribute that includes the data store name or other identification of an external data reference profile, (discussed below in reference to FIG. 6). A data store profile also includes connect information, and a specification of default cache rules. While the particular implementation uses a profile structure to store the data reference information, this is a matter of convenience. In practice, any available data structure may be used to store this information to meet the needs of a particular application.

Detailed Description Text (62):
A "Field" attribute and a "Query" attribute specify values used to query or retrieve specific data fields from the associated data store. Using an example where the external data store is a relational database, the Field=attribute specifies a value that is akin to the value of a "select" clause in an SQL query. Similarly, the Query=attribute specifies a value or values akin to the "from" and "where" clause in an SQL statement. It should be understood that the external database does not have to be a relational database, and the value of the field=and query=attributes is tailored to meet the needs of the associated data store.

Detailed Description Text (65):
Preferably, object 501 includes an attribute specifying cache rules. In the preferred implementation object 501 is instantiated in response to an attempt to evaluate attribute 402 shown in FIG. 4. Object 501 will return one or more data items retrieved from the associated external data store to the calling profile and these returned data items will be used as the value for that attribute. As a general rule the profile service instance 201 will not store or cache the returned data items in dedicated data store 206 or 207. The profile service instance does store the external reference in the dedicated data store, but not the data itself. However, in some applications is may be desirable to store a cache copy of the returned data items in a cache data structure such as data store cache (DS$) 325 shown in FIG. 3. A cache copy may allow core profile engine 301 to retrieve the external data more quickly if it is used again, and may enable core profile service to return data if a link to the specified external data store is not available for some reason.

Detailed Description Text (66):
Cache rule attributes that are specified include, for example, a value indicated whether or not to cache, a value indicated that if data is cached how long will it remain valid, and a value indicating whether an expired cached value may be used if a link to the external data store cannot be made to refresh the cached value. The DS$ available to the profile service instance can thus serve as a smart cache by automatically refreshing its contents according to the specified cache rules. This feature enables mission critical applications to continue operation with the appearance of failsafe operation when in fact the underlying virtual data store 314 is less reliable. While this feature goes against conventional wisdom of data store access, it should be recalled that the present invention is enhanced to store data that is seldom changed and is not primary data that is being used for data analysis and the like. Hence, the uncertainty with respect to the freshness of cached values is a tolerable uncertainty and a strong preference may be given in some instances to making sure that the client application requesting the profile data gets useful data in return.

Detailed Description Text (67):
FIG. 6 shows an example external data store profile 601 that in the particular embodiment comprises a profile stored in dedicated data store 206 and/or data store 207. The external data store profile 601 is a conventional profile object that names an external data store connector object (PhoenixLDAP_Datastore in FIG. 6). External data store profile 601 may include a number of optional properties that are passed to and used by the referenced external data store object. The referenced external datastore connector object is a predefined object from the class 731 shown in FIG. 7D. The external datastore object contains the data store-specific connection methods used to establish a link to an external data store. It is contemplated that the external data store connector class will be defined by the provider of the external

datastore mechanism as the class definition will extend the usefulness of their external datastore. Alternatively, the external datastore class may be defined by third party vendors or users.

Detailed Description Text (69):
The profile service supports two basic functional objects, profiles themselves and a "profile manager". Additionally, two support classes called external data reference (FIG. 7C) and external data store connector (FIG. 7D) define methods used specifically to access external data store mechanisms. The logical interfaces shown in FIG. 7A thorough FIG. 7D are not intended to be literal. Instead they are intended to articulate fundamental functional operations that the service supports. All implementations of the profile service desirably support these classes of functions to implement external data store links. In addition, individual implementations may support additional methods that are not supported in all implementations to meet the needs of a particular applications.

Detailed Description Text (70):
FIG. 7A lists functions implemented in profile objects. All of the listed functions require the specification of a profile upon which the function will operate. The profile can be specified, for example, by passing context information from the requesting entity to the profile service in the request message. The profile class shown in FIG. 7A lists functions available in instances of profile objects. In general, this category of methods manipulate attributes within a specified profile. Hence, once a profile object is created it is autonomous in the sense that it can be directly accessed by user calls and it no longer relies on the profile manager (discussed in reference to FIG. 7B) to enable attribute manipulation.

Detailed Description Text (72):
Schema methods within profile objects create and maintain a profile schema. A profile schema is created to enforce specified properties on all profile instances of a particular type. For example, consider a profile of type=employee. If no schema is defined, for each instance of the type=employee an independent profile is created in such a way that each profile can have a different set of attributes, subprofile bindings, and external data references. In contrast, when a schema is defined for a specified profile type, the schema specified a minimum set of attributes that must be included in the new profiles of that type and enforced upon new instances of that

Detailed Description Text (76):
The profile manager class 711 shown in FIG. 7b provides a mechanism for creating, retrieving and establishing schemas for profiles. Essentially, the methods summarized in FIG. 7B include factory methods that create new profiles (including entity profiles and profile lists), retrieve profiles, search profiles, define profile schema, and implement external data store methods. Search methods are substantially similar to the search methods used in profile objects, however, search all attributes within a specified profile rather than all attributes hierarchically below a specified profile.

Detailed Description Text (77):
Of particular interest to the present invention are external data store methods included in the profile manager class used to manipulate external data store profiles and external data reference objects. The "defineExternalDatastore" method is used to define a new external data store profile such as profile 601 shown in FIG. 6. This method accepts as parameters a name which comprises, for example, a string value specifying an arbitrary or user-defined name for the new external datastore. The defineExternalDatastore method also accepts a string value indicating the name of a datastore connector class 731 (shown in FIG. 7D) that will implement the external datastore object for this specific data store.

Detailed Description Text (79):
The "removeExternalDatastore" method removes or deletes an already defined external data store profile from the dedicated datastore 206 and/or 207. This method may also use the client context parameter to verify identity and privileges and returns a boolean "True" if the remove operation is completed successfully. The "ListExternalDataStores" method returns a listing of external datastore profiles defined within a given profile service instance. The steps followed to create an external data link using the methods shown in FIG. 7B are described in greater detail with reference to the flow diagram shown in FIG. 8.

Detailed Description Text (80):
The "newExternalDataReference" method creates a new external data reference object such as external data reference object 721 shown in FIG. 7C. The newly defined external data reference object can then be assigned to the value of any attribute in any profile. In a particular example the newExternalDataReference method accepts as a parameter a string indicating the ID of the external data store profile (e.g., profile 601) that should be used to gain access to the true data store for this value.

Detailed Description Text (81):
An instance of a new external data reference object encapsulates the information items shown in FIG. 5 as well as implementing the behavior and interface defined by the methods shown in FIG. 7C. Other string parameters specify initial values for the field, query, sample profile, and cache rules (as shown in FIG. 5) for the new external data reference object. The client context information may be used to verify the calling client's identity and privilege to perform the operation. The newExternalDataReference method returns a boolean value indicating success or failure of the method. An external data reference object 721 returns a value to an attribute that names the external data reference object. The returned value comprises one or more pieces of information extracted from an external datastore named within the external data reference object 721.

Detailed Description Text (82):
External data reference objects 721 include methods to set and get the attribute values within that object's data items (shown in FIG. 5). When an external data reference object is created (using the newExternalDataReference method) it is assigned a value for the DataStoreProfile attribute. Subsequently, a data store reference object can be queried to return the value of this attribute. The functionality of other methods within external data reference object 721 set and get the corresponding information items shown in FIG. 5.

Detailed Description Text (83):
The external data reference object defines an interface that interacts with a serialized interface of an external datastore connector object 731 shown in FIG. 7D. This serializable interface is used to communicate connection and access requests made by an external datastore connector over a network link. A data store connector object 731 will have an interface that accepts instances of the external data reference object 721 as a parameter. The data store connector object 731 returns data records to the external data reference object 721. In turn, external data reference object 721 returns a value (which may comprise multiple data items) to the profile in which the external data reference object was evaluated.

Detailed Description Text (84):
Important methods within an external datastore connector object include an init( ) method that instantiates the external datastore object. The init( ) method may be called by the external data reference object 721 or by other mechanisms within the profile service to test or verify a connection to an external data store. The connect( ) method establishes a logical connection to the external data store mechanism and may use any protocol supported by the external data store mechanism including hypertext transport protocol (HTTP), fibre channel protocols, common object request broker (CORBA), remote method invocation (RMI) and the like. The reconnect( ) method operates in a similar fashion to reestablish a previously established connection. In a particular implementation the connect( ) method is called automatically when a connector object is instantiated.

Detailed Description Text (85):
The read( ) and write( ) methods take a query string provided by, for example, an external data reference object 721 and perform a read or write (update) operations on the specific data record(s) specified by the query string. A read( ) method returns an arbitrary number of records generated by the external data store. A write( ) method returns a boolean value indicating success of the update operation. External data store connector object 731 also includes a validateReference( ) method that accepts a candidate query and returns a boolean value indicating whether the candidate queyr will resolve to real data. An query can be tested by instantiating an external data reference object 721 with sample data from the specified sample profile and passing the external data reference object 721 to the external data store connector object 731 with a call to the validateReference( ) method. The optional close( ) method is used to expressly close a connection to an external data store although it is contemplated that a service may operate without a need to expressly close a connection.

Detailed Description Text (86):
FIG. 8 shows steps in a process of creating the data structures that will implement a link between a service application and an external datastore. In step 801 the process in accordance with the present invention is started by creating a datastore connector class that defines data store-specific connection methods and parameters. Step 801 may be performed by a data store manufacturer or provider, a user with detailed knowledge of the external data store connection protocol(s), or a third party software vendor. Each external data store mechanism such as devices 208, 209, 210, 216 and 217 shown in FIG. 2, will require its own datastore connector class definition and corresponding datastore object(s).

Detailed Description Text (87):
A client application creates an external data store profile in step 802 using, for example, the defineExternalDataStore( ) method shown in FIG. 7B. Step 802 is performed by passing the required parameters to the defineExternalDataStore( ) method resulting in an external datastore profile being created in dedicated datastore 206 and/or 207. In step 803 an external data reference object is created by calling to, for example, the newExternalDataReference( ) method shown in FIG. 7B. In the particular example step 803 results in a profile like that shown in FIG. 5 being stored in the dedicated datastore.

Detailed Description Text (89):
FIG. 9 shows steps in a process of using the data structures created by the steps shown in FIG. 8 that will establish and use a link between a service application and an external datastore. When an attribute having a value that is an external data reference object is being evaluated, named external data reference object is instantiated in step 901. Step 901 uses parameter data passed to the external data reference object at runtime to fill any parameterized values specified in the field or query data items. The external data reference object reads the external datastore profile in step 902 to determine, among other things, the identity of the datastore connector object it must use to actually link to the external data store.

Detailed Description Text (90):
The external data reference object instantiates the specified connector object in step 903 by calling an init( ) . Preferably step 903 results in a connection being established by, for example, a connect( ) method within the connector object. The external data reference object is passed as a parameter to the connector object in step 904. The connector uses the received parameter to issue a query on the external data reference mechanism in step 905. This query may be a read or write query depending on the needs of the particular application. In step 906 the data returned by the external data store is returned to the external data reference object and supplied as the data attribute's value in step 907.

Current US Original Classification (1):
707/4

Current US Cross Reference Classification (1):
707/100

Current US Cross Reference Classification (2):
707/103R

Current US Cross Reference Classification (3):
707/2

Current US Cross Reference Classification (4):
707/200

Current US Cross Reference Classification (5):
707/3

CLAIMS:

1. A mechanism for linking an external data store to a profile service instance, the mechanism comprising:

a profile service instance having a number of predefined built-in functions and a number of profile object instances;

an external data store interface within the profile service;

a data store connector object instance having methods therein for managing a connection to a specified data store; and

a data store query object having a name, the data store query object including parameter data therein for selecting data from the data store specified by the data store connector object.

2. The mechanism of claim 1 wherein the data store query object is instantiated in response to evaluating the profile object instances that references the data store query object's name.

3. The mechanism of claim 1 wherein each profile instance comprises a plurality of attributes where a value of at least one attribute refers to the data store query object.

4. The mechanism of claim 1 further comprising an external data store definition having a name that is referenced by the data store query object and an attribute indicating the data store connector object instance, wherein the data store connector definition is stored as one of the plurality of profile objects.

5. The mechanism of claim 1 wherein the methods contained within the data store connector object include a method to create a runtime link between the profile service and the specified data store.

6. The mechanism of claim 1 wherein the methods contained within the data store connector include a method to read data from the specified data store using the parameter data contained in the database query object.

7. The mechanism of claim 1 wherein the methods contained within the data store connector include a method to write data to the specified data store using the parameter data contained in the database query object.

8. The mechanism of claim 1 further comprising a dedicated data store coupled to the profile service instance, the dedicated data store comprising structures for storing the profile object instances;

a cache structure within the dedicated data store; and

cache method definitions stored within the data store query object specifying a manner in which data returned from the specified external data store is cached in the cache structure.

9. The mechanism of claim 8 wherein the data store query object further comprises methods for accessing selected data from the cache rather than from the data store connector object.

10. The mechanism of claim 4 wherein the parameter data in the data store query object specifies a data store name matching the data store name of the external data store definition, a field list indicating fields from the external data store to be included, and a query string indicating portions of the external data store that will participate.

11. The mechanism of claim 10 wherein the parameter data includes parameterized values that serve as placeholders for parameters supplied by the profile service instance upon instantiation of the data store query object.

12. The mechanism of claim 11 further comprising:

a sample profile within the data store query object, wherein the sample profile includes attributes specifying sample values corresponding to the parameterized values; and

a validation method in the connector object that tests validity of the external data store profile using the sample values in the sample profile.

13. A mechanism for creating an external data link to a profile service instance, the mechanism comprising:

a profile service instance having a number of predefined built-in functions and a number of profile object instances;

a dedicated data store coupled to and exclusively accessed by the profile service instance;

a data store-specific external data store connector object definition accessible by the profile service instance and defining a plurality of methods for connecting to an external data store;

an external data store reference class accessible by the profile service instance and defining and external data reference object having a plurality of methods for manipulating attributes within instances of the data reference class;

a method within the profile service instance operable to create an external data store profile in the dedicated data store, where the external data store profile has a name and specifies the connector object definition, and a number of data store-specific properties;

a method within the profile service instance operable instantiate an external data reference object based upon the external data reference class, where the external data reference object specifies a data store name matching the data store name of the external data store profile, and a field list indicating fields from the external data store to be included;

a method within at least one profile instance operable to set a value of an attribute within that profile instance to the name of the external data store profile.

15. The mechanism of claim 13 wherein evaluation of a profile attribute having a value equal to the name of an external data store causes the profile service to instantiate an external data store connector based upon the external data store profile, the external data store instance having a plurality of methods for accessing the external data store named by the external data store profile using the connector specified by the external data store profile.

16. The mechanism of claim 15 further comprising an attribute within the external data reference profile that specifies whether external data returned by the external data store object is cached in the dedicated data store.

17. The mechanism of claim 13 where the query string includes parameterized values that serve as placeholders for values supplied by an external data store instance.

18. The mechanism of claim 17 wherein the external data reference profile includes a sample profile, wherein the sample profile includes attributes specifying sample values corresponding to the parameterized values; and

a validation method in the external data store instance that tests validity of the external data store profile using the sample values in the sample profile.

20. A method for linking a to an external database comprising the steps of:

providing a data store connector class defining an interface and methods for connecting to and accessing data from a specified external data store;

defining an external data store profile having a name and comprising a plurality of attributes, wherein at least one attribute specifies the data store connector class and at least one attribute specifies a parameter used by instances of the data store connector class;

creating an external data reference object having a name, a plurality of parameter strings, and a reference to the external data store profile; and

creating a profile comprising a plurality of attributes where the value of at least one attribute is set to the name of the external data reference object.

21. The method of claim 20 wherein the step of providing a connector class further comprises:

defining a data store connect module having an interface and methods for managing a connection to a specific external data store; and

defining a data store query module having an interface and including methods to access the external data store specified by the connect module.

22. The method of claim 20 wherein instances of the data store connector class have an interface that accepts instances of the external data reference object as a parameter.

23. The method of claim 20 further comprising the step of:

including a sample profile in the external data reference object;

instantiating the external data reference object with parameters from the sample profile;

validating the external data reference object by passing the external data reference object as instantiated with the sample profile parameters to an instance of the data store connector class.

24. A distributed profile service system for providing profile services to a number of user software applications, the profile service system comprising:

a plurality of geographically distributed profile service instances, each profile service instance being coupled to a dedicated data store;

an external data store interface within each profile service instance;

a plurality of external data store mechanisms;

an external data store connector object coupled to at least one profile service instance, the data store connector object having methods therein for managing a connection to a specified data store;

an external data store connect profile stored in the dedicated data store of the at least one profile service instance, the connect profile naming the external data store connector module;

a data store reference object having a name, the data store reference object including parameter data therein for selecting data from the data store specified by the data store connect profile; and

a profile within the at least one profile service instance that includes an attribute naming the data reference object.

25. A computer program product embodied in a tangible form comprising:

computer program devices readable by a data processor coupled to receive the propagating signal for managing a profile data service, the computer program product comprising:

first program code devices providing a data store connector class defining an interface and methods for connecting to and accessing data from a specified external data store;:

second computer program code devices configured to cause a computer to define an external data store profile having a name and comprising a plurality of attributes, wherein at least one attribute specifies the data store connector class and at least one attribute specifies a parameter used by instances of the data store connector class;

third computer code devices configured to cause a computer to create an external data reference object having a name, a plurality of parameter strings, and a reference to the external data store profile; and

third computer code devices configured to cause a computer to create a profile comprising a plurality of attributes where the value of at least one attribute is set to the name of the

external data reference object.

30. A mechanism for linking a to an external database comprising the steps of:

means for providing a data store connector class defining an interface and methods for connecting to and accessing data from a specified external data store;

means for defining an external data store profile having a name and comprising a plurality of attributes, wherein at least one attribute specifies the data store connector class and at least one attribute specifies a parameter used by instances of the data store connector class;

means for creating an external data reference object having a name, a plurality of parameter strings, and a reference to the external data store profile; and

means for creating a profile comprising a plurality of attributes where the value of at least one attribute is set to the name of the external data reference object.

Previous Doc Next Doc Go to Doc#

Freeform Search

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Term: L9 and (instance near type)

Display: 50 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Friday, March 04, 2005 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L10	L9 and (instance near type)	17	L10
L9	L8 and relationship	33	L9
L8	L7 and instance	34	L8
L7	L6 and schemas	34	L7
L6	L5 and synchroniz$	50	L6
L5	l4 and (file near system$)	103	L5
L4	L3 and (application near program$)	260	L4
L3	L2 and (data near store)	410	L3
L2	L1 and (database near engine)	929	L2
L1	707/$.ccls.	25377	L1

END OF SEARCH HISTORY